

**PREDICTIVE PREFETCHING TO IMPROVE
PARALLELIZATION OF DOCUMENT GENERATION SUBTASKS**

Background of the Invention

Field of the Invention

[0001] The present invention relates to techniques for reducing document generation times in systems that dynamically generate web pages or other electronic documents in response to user requests. More specifically, the invention relates to software architectures and predictive prefetching methods for increasing the degree to which document generation subtasks can be performed in parallel.

Description of the Related Art

[0002] The task of dynamically generating a requested electronic document, such as a web page, commonly involves the execution of multiple data retrieval subtasks. Each such subtask may, for example, involve the generation of a service request for content used to generate the requested document. To reduce document generation times, some or all of these data retrieval subtasks may, in some cases, be performed in parallel. In many cases, however, the data retrieval subtasks cannot be performed in parallel because one subtask is dependent upon the results of another.

[0003] For example, the task of generating a web page may involve sending an initial service request to a service that returns a variable list of items represented within a database. For each item in this list, a second service may then be called to request catalog content or other data associated with that item. Assuming the list of items can vary from one page rendering event to the next, the details of the calls to the second service cannot be accurately determined until the first service has returned the list. Consequently, using existing methods, the requests to the second service must be postponed until the initial service request is completed.

Summary of the Invention

[0004] In accordance with the present invention, the above and other problems are addressed by preemptively initiating selected data retrieval subtasks, such as selected service requests, to prefetch data that is deemed likely to be used to generate requested documents. Data retrieval subtasks that would otherwise be postponed pending the outcome of other subtasks are thereby performed in parallel with such other subtasks, reducing document generation times. Data retrieved unnecessarily as the result of preemptive subtask initiation may be ignored.

[0005] The subtasks to be performed preemptively in response to a given document request are preferably selected based at least in part on predictions of the subtasks that will be used to generate the requested document. These predictions may be based on historical data regarding the subtasks actually used to generate specific documents. For example, in one embodiment, historical data is used to build a mapping table that maps specific document generation tasks to corresponding sets of subtasks that have been performed previously as part of such tasks. The mapping table may, for example, map a particular dynamically generated web page to a set of subtasks that have been performed at least N% of the times the web page has been generated. When a particular document request is received from a user, this table is used to look up a set of one or more subtasks that are to be initiated immediately. Criteria other than subtask frequency-of-use may also be incorporated into the selection process.

[0006] In one embodiment, the mapping table is updated in real time to reflect the subtasks actually used to generate specific documents, such that the subtask “predictions” adapt automatically in real time. The mapping table may alternatively be updated or generated periodically based on an off-line analysis of collected data.

[0007] Neither this summary nor the following detailed description purports to define the invention. The invention is defined by the claims.

Brief Description of the Drawings

[0008] Figure 1 illustrates a web site system that uses predictive prefetching to dynamically generate web pages according to one embodiment of the invention.

[0009] Figure 2 illustrates one method that may be used to generate a task-to-subtask mapping table of the type shown in Figure 1.

[0010] Figure 3 illustrates a sequence of events that may occur when a web page is requested in the system of Figure 1.

Detailed Description of the Preferred Embodiments

[0011] For illustrative purposes only, the invention will be described largely in the context of a web site system that provides access to an electronic catalog of items represented in a database. The items may, for example, be physical products, journal articles, news articles, auctions, web sites, businesses, music files, or any other type of item that can be represented within a database. As will be recognized, the invention is also applicable to other types of dynamic web page applications. In addition, the invention may be applied to the dynamic generation of other types of electronic documents, such as voiceXML documents and email documents.

I. Overview

[0012] Figure 1 illustrates a web site system 30 according to one embodiment of the invention. The system 30 includes one or more web server machines 32 (one shown) that respond to requests from user computers 34 by generating and returning web pages. Each such web server machine (“web server”) executes a dynamic page generator program 36 that generates the web pages in response to such requests (i.e., the web pages are generated dynamically).

[0013] The dynamic page generator 36 preferably generates the web pages according to web page templates stored in a database 40, as is known in the art. For example, an item detail page template may be used to generate item detail pages for each of the catalog items represented in a database. Similarly, a browse node template may be used to generate pages that correspond to specific nodes of a hierarchical browse structure in which the catalog items are arranged by category. Many different templates may, of course, be provided within a given system 30. Although templates are used in the preferred embodiment, the invention does not require the use of templates.

[0014] As depicted in Figure 1, the dynamic page generator 36 includes a template processor 42 that is responsible for processing the web page templates, and for generating service requests specified by such templates. When a page request is received by the web server 32, the template processor 42 retrieves and processes the corresponding template, which may be uniquely identified by the target URL (Uniform Resource Locator) included within the page request. Typically, this template will specify one or more service requests that are to be sent to specific services 44 to retrieve content for generating the requested page. These services 44 are referred to herein as “content services,” as they return data that is included within, or that otherwise affects the content of, the requested web pages.

[0015] To facilitate parallel execution of the service requests associated with a given page request, the template processor 42 preferably uses a linear thread to sequentially process each web page template, and uses an event-driven thread to generate and wait for the results of the associated service requests. Other software architectures that support the parallel execution of page generation tasks may alternatively be used.

[0016] As illustrated in Figure 1, each content service 44 is typically in the form of a service application 44A that runs on one or more physical servers 44B. Many tens or hundreds of different content services 44 may be provided within a given system 30. The types and functions of these services 44 may vary widely depending on the nature and purpose of the particular web site system 30 (e.g., business-to-consumer sales, access to online publications, Internet searching, job postings, online banking, etc.).

[0017] In many cases, the input parameter(s) or other details of a given service request remain unknown during template processing until the results of one or more other service requests are completed. To provide a simple example, suppose that each item detail page generated by the web site system 30 includes a “similar items” section that displays short descriptions of three items that are deemed to be similar to the item featured on the page. To generate the similar items section, a request is initially sent to a Similar Items service 44 to obtain the IDs of three items that are similar to the featured item. The IDs of these three similar items are then used as input parameters of respective service calls to a Catalog service 44, which responds by returning short item descriptions that are incorporated into the requested page. Assume also that the Similar Items service 44 uses an algorithm that

provides a degree of variability, such that the three similar items listed on a particular item detail page may change over time in response to user activity, changes to the catalog, and/or other factors.

[0018] If the requests to the Catalog service 44 in this example are blocked or otherwise postponed until the Similar Items service 44 returns the list of similar items, the page generation time will be adversely affected. This problem is compounded when a greater level of service request dependency or “chaining” exists. For instance, a first service request may return a result that is used to generate a second service request, and the result of the second service request may then be used as a parameter of a third service request, and so on. If these chained service requests are performed sequentially rather than in parallel, the time required to generate the page will be at least as long as the sum of the response times of the individual service requests.

[0019] The preferred embodiment of the present invention addresses this problem by using a predictive prefetch service 50 to identify those service requests that are likely to be made as part of a page generation task. The predictive prefetch service 50 (hereinafter “prefetch service”) preferably makes these predictions based on statistical data regarding service requests made during prior renderings of the requested web page and/or related web pages. The service requests identified by the prefetch service 50 as being probable are then made “preemptively” at or near the outset of the page generation task, rather waiting for requests to other content services 44 to be completed. If a given service request is made unnecessarily (i.e., the prediction is wrong), the associated data returned by the particular content service 44 is ignored. Data regarding of the accuracy of the predictions is collected over time by monitoring the dynamic page generation process, and is used to adaptively modify the predictions.

[0020] For instance, assume that a user request is received for an item detail page for item 12345, and that item detail pages are generated as described above. The prefetch service 50 may access a repository of historical/statistical data to determine that requests for the item detail page for item 12345 have resulted in service requests being made as follows, disregarding any service requests made unnecessarily due to prefetching:

<u>Service: Request</u>	<u>% of time made</u>
Catalog: full description of item 12345	100
Similar Items: items similar to item 12345	100
Catalog: short description of item 23456	96
Catalog: short description of item 34567	93
Catalog: short description of item 45678	87
Catalog: short description of item 56789	82
Catalog : short description of item 67890	25

[0021] As described below, this statistical data may be derived from a particular set of activity data, such as activity data collected over the last M days or weeks. If a frequency threshold of 80% is used to select the service requests that are to be made preemptively, the first six of these service requests will be made at or shortly after the outset of the page generation task. The item detail page will ordinarily be generated in less time than would be required if the requests for the short descriptions of the similar items were postponed until completion of the Similar Items service request.

[0022] Service requests are one example of a type of document generation subtask that can be performed predictively/preemptively in accordance with the invention. The prefetch service 50 can also be used to identify, and preemptively perform, other types of document generation subtasks, including but not limited to other types of data retrieval subtasks. For instance, predictive prefetching may be used to identify, and preemptively perform, database lookup operations that do not involve the generation of a service request.

[0023] The data retrieval subtasks that are performed preemptively in accordance with the invention may involve the retrieval of static data and/or dynamically generated data. The retrieved data may be data that is not actually incorporated into the requested web page or other document. For example, a service request may be issued preemptively for purposes of confirming that the user is authorized to access a particular resource, or to initiate validation of a credit card; in both of these examples, the retrieved data may be a status value indicating the outcome of the service request.

[0024] Figure 1 illustrates the components and the basic operation of the prefetch service 50. The service 50 is implemented as a prefetch service application 50A that runs on a prefetch server 50B. The prefetch service application 50A may alternatively run on the web server machine(s) 32, or on a physical server that is also used to implement one or more other services. Although a single prefetch server 50B is shown, the prefetch service application 50A may be replicated across multiple physical servers to accommodate large numbers of concurrent page requests.

[0025] As illustrated, the prefetch service application 50A accesses a task-to-subtask mapping table 52 that maps tasks to corresponding subtasks. This mapping table 52 is used to identify the set of service requests, or other subtasks, that are to be performed preemptively for a given page generation task. Each entry in the mapping table 52 includes a task identifier portion 52A that describes a particular task. As discussed below, a given task identifier may, for example, correspond to a particular URL, or to a collection of URLs that point to a common web page or web page template. The task-to-subtask mapping table 52 may, for example, be stored as a hash table or a B-tree. As will be recognized, however, the mapping table 52 can be implemented using any type of data structure or set of data structures capable of mapping tasks to subtasks.

[0026] Each table entry also includes an associated “subtask profile” portion 52B that specifies associated subtasks that have been performed frequently as part of the identified task. For example, the subtask profile 52B for a given task may specify all of the service requests that are properly made at least 80% of the time that task is performed. In the embodiment depicted in Figure 1, each subtask is listed together with a score value (shown in parenthesis) that generally represents the importance of performing that particular subtask preemptively.

[0027] In one embodiment, each such score value is a frequency-of-use score reflecting the frequency with which the respective subtask has been performed as part of the corresponding task over a defined time period. For example, assuming frequency-of-use scores are used, the table 52 in Figure 1 indicates generally that subtask_1 has been performed 90% of the time task_1 has been performed, excluding unnecessary occurrences of subtask_1 caused by predictive prefetching. A subtask which is performed unnecessarily as

the result of prefetching is not considered as being performed “as part of” the task, and is referred to herein as being “unused.” Such unnecessarily performed subtasks are not counted when generating the frequency-of-use scores, so that the scores accurately reflect actual subtask usage. In this example, each score value generally represents the probability that the particular subtask will be used as part of the corresponding task.

[0028] The scores may optionally be generated so as to give greater weight to the most recent occurrences of the relevant page generation tasks than to earlier page generation events. In addition, other criteria may be taken into consideration in generating the scores, such as the average execution time of the particular subtask, and/or the average point at which the subtask is ordinarily performed within the overall page generation event (e.g., at the outset versus near the end of the page generation task). Although score values are depicted in Figure 1, they may alternatively be omitted from the table 52, or may be generated on-the-fly from other data values stored in the table 52.

[0029] Subtasks that occur only at the outset of a given page generation task may be omitted from the relevant table entry since there is little or no benefit to initiating such subtasks preemptively.

[0030] The dynamic page generator component that communicates with the prefetch service 50 is referred to herein as the prefetch client 56. As described below, the prefetch client 56 is responsive to a URL/page request by querying the prefetch service 50 for a list of the service requests (subtasks) that are likely to be performed during generation of the requested page. Upon obtaining this list, the prefetch client 56 or another component of the dynamic page generator 36 determines whether the template processor 42 has already made one or more of the specified service requests, and to the extent it has not, makes these service requests preemptively. If the template processor 42 thereafter attempts to make a service request that has already been made preemptively, the redundant service request is ignored or is otherwise prevented from being sent.

[0031] Once the page generation task is complete, the prefetch client 56 may send an “end-of-page feedback” message to the prefetch service 50 specifying some or all of the service requests that were actually used to generate the requested page. This allows the prefetch service 50 to monitor the accuracy of its predictions. As described below, the

prefetch service preferably uses such feedback messages to update the task-to-subtask mapping table—in real time and/or on a periodic basis. By updating this table 52 based on the feedback messages, the prefetch service 50 adaptively modifies its predictions of the service requests (subtasks) that will be performed.

[0032] As illustrated in Figure 1, the prefetch client 58 may access a prefetch cache 58, which may be implemented with random access memory and/or hard disk storage of the web server machine(s) 32. The prefetch cache 58 is preferably used to cache data retrieved from the prefetch service 50 in order to reduce the frequency with which queries are sent to the prefetch service. For example, the cache 58 may store the subtask profile data for some or all of the most frequently accessed web pages, so that requests for these pages can be processed on most page requests without querying the prefetch service 50. Each entry in the prefetch cache 58 may correspond to a particular entry in the task-to-subtask mapping table 52, and may be created or updated based on data returned by the prefetch service. In one embodiment, the prefetch service 50 assigns a time-to-live (TTL) value to each such cache entry to specify how long the associated entry may be used. The TTL value for a given task ID may be generated based on collected historical data indicative, for example, of how often new subtasks are added to the associated subtask profile 52B.

[0033] The system 30 may, if necessary, be configured to prevent certain types of service requests from being performed preemptively. For example, the system may be prevented from preemptively issuing service requests that cause database updates to occur. This may be accomplished, for example, by configuring the prefetch client 56 and/or the prefetch service 50 to allow prefetching only for services 44 identified in an “opt-in” list.

II. Task-to-subtask mapping table

[0034] For simple web site systems, a separate entry may be provided in the mapping table 52 for every possible URL that may be requested by a user. In more complex systems, some or all of the task identifiers 52A may be sufficiently broad to encompass multiple possible URLs that point to the same web page or to a set of related web pages (such as those that correspond to the same template). For example, the task identifier [page type = browse node; node ID = 5ACFD] may encompass all URLs that point to the browse node page for node 5ACFD. Thus, although a given URL request for this browse node page may

include various parameters that specify, for example, the state of the user's browsing session, these parameters may be ignored for purposes of looking up the associated subtasks to be performed. To provide another example, the task identifier [page type = browse node, node ID = ?] may encompass all browse node pages, all of which may share a common template.

[0035] Further, a degree of overlap may be provided between the task identifiers 52A included within the mapping table 52. For example, one table entry may be provided for each of the most frequently accessed item detail pages, while another table entry may be provided that is generic to all item detail pages. The generic "item detail page" entry would specify those subtasks, if any, that are frequently performed for all item detail pages, and may be used if no table entry exists for the particular item detail page being requested. For example, assume that a particular catalog item is a new addition to the catalog, or is accessed very infrequently. In either case, it may not be practical or desirable to include a separate table entry for this item's detail page. When the item's detail page is requested, the generic "item detail page" entry may therefore be used to identify any service requests or other subtasks that are to be performed preemptively. Assuming certain subtasks are commonly performed for all item detail pages, an increased level of subtask parallelization may be realized in this example over the case in which no prefetching is performed.

[0036] In the embodiment depicted in Figure 1, the task-to-subtask mapping table 52 preferably serves as a repository of historical data regarding service requests made, or other subtasks performed, for specific tasks. For example, for each task identifier 52A, the mapping table 52 may store a listing of some or all of the subtasks that have historically been performed as part of the corresponding task, including the input parameters of such subtasks. Each such subtask may be listed in the subtask profile portion 52B of the table entry together with some or all of the following statistical data: (a) the number of times that particular subtask has been performed since inception or over a defined time period, excluding "unused" subtasks performed due to prefetching (note that this number can be divided by the total number of occurrences of the relevant task to obtain a frequency-of-use score or ratio); (b) timestamps indicating the dates and times of occurrence of the subtask; (c) an average execution time of the subtask, and (d) the average amount of time between initiation of the task and initiation of the subtask. This data, which may be obtained from feedback messages

from the prefetch client 56, may be used to identify the subtasks that should be performed preemptively when a particular web page is requested. The timestamp data may also be used to effectively remove old feedback data from the mapping table 52, or to otherwise give greater weight to recent feedback data over old feedback data.

[0037] For example, in one implementation of the system of Figure 1, the prefetch service 50 maintains an “occurrences” count for each task and each associated subtask represented in the table 52. Periodically, all of the occurrences counts for a given task are proportionally reduced, such as by dividing the count values by ten or by one hundred. This reduces the effect of old feedback data while preserving the basic information regarding the relative frequencies with which specific subtasks are performed.

[0038] In another embodiment, which is depicted in Figure 2, the feedback data regarding the actual service requests (subtasks) used to render specific web pages is stored in a separate database or other repository 60, rather than in the mapping table 52 itself. An analysis component 62 periodically analyzes a most recent set of the feedback data stored in this repository 60 to generate a replacement task-to-subtask mapping table 52, or to make updates to an existing version of this table 52. As part of this process, the analysis component 62 may identify those tasks for which a sufficient quantity of feedback data exists, and for which a sufficient degree of subtask consistency/predictability exists, to justify the creation of a table entry. Once these tasks have been identified, the analysis component 62 may use a subtask scoring algorithm to select those service requests or other subtasks that are to be included in the associated table entry.

[0039] In this embodiment of Figure 2, the subtask profile portion 52B of a given table entry may simply list the subtasks that should be performed preemptively, without listing any associated statistical data. Subtask score values may, but need not, be included in such a table 52. The collected feedback/activity data may be stored in a log format as shown, although other formats, including preprocessed formats, may be used.

[0040] Referring again to Figure 1, some of the subtask descriptions included in the table 52 may be “tokenized” descriptions in which actual input parameters are replaced by tokens. For example, the subtask description “Recommendations <USERID>” may be used to specify to the prefetch client 56 that a service called Recommendations should be called to

obtain personalized recommendations for the particular user who is accessing the relevant page. The prefetch client 56 would recognize the string “USERID” as a token in this example, and replace it with the actual user ID for purposes of calling the Recommendations service. One example of how input parameter tokenization may be used is provided in section VI below.

III. Example Process Flow

[0041] Figure 3 illustrates the general sequence of interactions that occur, in one embodiment, between the dynamic page generator 36 and the prefetch service 50 in response to a web page request. This example assumes that the subtask profile data for the particular page generation task does not reside in the prefetch cache 58 and must therefore be retrieved from the prefetch service 50. The dynamic page generator 36 may send service requests to the content services 44 (as depicted by the dashed arrow) any time between events 2 and 6 of Figure 3.

[0042] When a page request is initially received (event 1), the prefetch client 56 immediately intercepts the page request and, assuming the relevant subtask data does not reside in the prefetch cache 58, sends a prefetch query message (event 2) to the prefetch service 50. As illustrated, the prefetch query message is sent together with a description of the associated page generation task, as described above. The task description may, for example, consist of the URL of the requested web page (“target URL”), or may consist of information extracted from the target URL.

[0043] In addition, the task description sent with the prefetch query may, in some implementations, include parameters that are not part of the target URL. For example, the task description may include information about the user (e.g., recognized versus unrecognized), form data entered by the user, and/or state information descriptive of the user’s browsing session. As with information contained within the URL, these parameters may be used to select the appropriate entry in the task-to-subtask mapping table 52.

[0044] For instance, if the page request (event 1) is a submission of a search query for conducting a general World Wide Web search, the search query itself may be passed to the prefetch service 50 and used to look up the appropriate set of subtasks. These subtasks may, for example, involve service requests for retrieving information about specific search

result URLs/web pages. In this example, the task-to-subtask mapping table may include a separate entry for each of the most frequently submitted search queries.

[0045] As depicted by event 3 in Figure 3, the prefetch service 50 responds to the prefetch query by looking up from the task-to-subtask mapping table 52 a listing of the subtasks that can be performed preemptively. As part of this process, the prefetch service 50 preferably maps the received task description to the most specific task ID 52A to which it corresponds. For instance, if the task description is the URL of a particular item detail page, the prefetch service may initially check to see if a table entry exists for this item detail page. If no such table entry exists, the service may fall back to a less specific table entry, such as one that encompasses all item detail pages. The table lookup operation can typically be performed relatively quickly in comparison to the overall page generation time. If no entry is found for the particular task description, the prefetch service may return a null response message.

[0046] In one embodiment, the query may optionally specify a score threshold, such as a minimum frequency-of-use score, for purposes of looking up subtasks from the table 52. In such cases, the prefetch service 50 looks up and returns a listing of those subtasks that meet or exceed this minimum score value. A fixed minimum score value may alternatively be used for all queries. Further, the minimum frequency-of-use score may be increased automatically when the content services 44 are heavily loaded (to reduce unnecessary service requests), and decreased when the content services are lightly loaded.

[0047] As depicted by event 4 in Figure 3, assuming a table entry is found, the prefetch service 50 returns a response message that includes a listing of subtasks, such as those that satisfy the specified score threshold. The response message may also indicate whether this subtask data should be stored in the prefetch cache 58. The prefetch service 50 (or alternatively the prefetch client 56) may decide whether the subtask data for a particular task is to be cached based on collected data regarding the frequency of occurrence of that task, and possibly other criteria.

[0048] The response message (event 4) may also include information specifying whether, or how often, the prefetch client 56 should return end-of-page feedback messages for this particular task. For example, in one embodiment, the prefetch service 50 may

indicate that feedback should be provided with a probability $1/N$ (e.g., $1/100$), in which case the prefetch client will have a $1/N$ chance of providing feedback for this particular instance of the task. This feature may be used to inhibit the transmission of feedback messages for tasks that are very frequently performed, as may be desirable for reducing the load on the prefetch service 50. For instance, if a particular page is requested several times per second on the average, the prefetch service 50 will accumulate sufficient subtask data for this page even if feedback is provided for only a small fraction of the page generation events. The prefetch service 50 may select the feedback probability value for a particular task, or otherwise control the frequency with which feedback is provided for that task, based on collected data regarding the frequency of occurrence of that task, and possibly other criteria.

[0049] As depicted by event 5 in Figure 3, the dynamic page generator 36 (and more specifically, the prefetch client 56) then initiates the subtasks specified by the prefetch service's response message, except for any such subtasks that have already been initiated by the template processor 42. Subtasks initiated preemptively by the prefetch client 56 are initiated on behalf of the template processor 42, before it is known whether the template processor will actually attempt to perform such subtasks. As mentioned above, each such subtask is in the form of a service request in the preferred embodiment, although other types of subtasks may be supported. Data retrieval subtasks initiated as the result of the prefetch service represent predictive prefetch events, as data is being retrieved based on a prediction that it will be used to generate the requested web page. As mentioned above, if the template processor 42 thereafter attempts to perform a subtask that has already been initiated preemptively, the dynamic page generator 36 may prevent the duplicate service request from being sent to the prefetch service 50.

[0050] If the subtasks that are to be performed preemptively include one or more service requests to the same content service 44, these service requests may optionally be combined or "batched" for sending to the relevant content service 44. For example, if it is determined that an Image service should be preemptively called twenty times to retrieve twenty separate images, these twenty service requests may be combined into a single service request to be sent to the Image service. The Image service may then return a single response

that contains all twenty images. The use of service request batching in this manner reduces network traffic and conserves processing resources.

[0051] The service request responses received from the content services 44 are used by the template processor 42 to construct the requested web page. The web page may be constructed throughout events 2-5 of Figure 3, and is returned to the user (event 6) upon completion.

[0052] In addition to subtask frequency-of-use data, various other criteria may be used in deciding whether a given subtask should be performed preemptively as part of a page generation task. For example, if a particular subtask takes much longer than other subtasks on the average, or is typically performed at the very end of a page generation task, it may be desirable to initiate this subtask preemptively even though its likelihood of use is only 50%. Further, the decision of whether to make a given service request preemptively may take into consideration the current load on the relevant service, so that unnecessary service requests are reduced or avoided when the service is heavily loaded.

[0053] In event 7, the dynamic page generator 36 sends a feedback message to the prefetch service 50 (assuming in this example that feedback is sent for this particular page request). This message preferably specifies the subtasks that were actually used to generate the web page. Other feedback data, such as the execution times of the service requests, may also be included in the feedback message. As mentioned above, the prefetch service 50 may inhibit or reduce the transmission of feedback messages if, for example, sufficient historical data already exists for the particular task. Feedback data regarding subtasks performed at the outset of the page generation task may optionally be excluded from the feedback messages.

[0054] In event 8, the prefetch service 50 updates the mapping table 52 to reflect the feedback provided by the dynamic page generator 36. As part of this process, the prefetch service 50 may update the score values associated with specific subtasks, and/or update data values used to generate such scores. Rather than performing these table updates in real time, a number of feedback messages can be aggregated, such that updates to the table are performed in batches.

[0055] If the prefetch service 50 is replicated across multiple servers 52B, each such server can update its own respective task-to-subtask mapping table 52 based on the

feedback it receives. Because each such server 52B will receive similar feedback data over time for like tasks, these tables will evolve similarly over time, and will thus contain similar data at any given point of time. As a result, each instance of the prefetch service 50 can operate independently of other instances. For web pages that are accessed infrequently (e.g., once per hour), the prefetch service 50 may instruct the prefetch client 56 to send the feedback data to all of the prefetch servers 50B, such that each such server may update its respective table 52.

[0056] As described above with reference to Figure 2, the feedback data can alternatively be maintained in a separate repository 60, and analyzed periodically by an analysis component 62 for purposes of generating the table 52. In such embodiments, each prefetch server 52 may use the table data most recently generated by the analysis component 62, without performing any table updates. Hybrid approaches are also possible in which some types of table updates are made in real time, and others are made based on an offline analysis of collected data.

[0057] Referring again to Figure 1, although the use of a separate prefetch service 50 is preferred, it will be recognized that some or all of the functionality of this service can alternatively be implemented by the prefetch client 56 on the web server machine 32. For example, rather than sending a prefetch query to the prefetch service 50 when a page request is received, the prefetch client could look up a listing of the associated subtasks from a locally-stored mapping table 52. This locally-stored mapping table could be generated periodically by an off-line analysis component, as described above.

IV. Requesting subtask data for both specific and generic task IDs

[0058] One variation to the process described above is to allow the prefetch client 56 to request the subtask profile data for both the specific task ID and the generic task ID for a particular task. For example, when a page request is received for a relatively infrequently accessed item detail page, the prefetch client 56 may generate two prefetch service requests: one for the subtask profile 52B of the requested item detail page, and one for the subtask profile of the generic item detail page. In this example, the subtask data for the generic item detail page would much more likely reside in the prefetch cache 58, resulting in a cache hit.

Consequently, the prefetch client 56 will be able to very rapidly identify, and preemptively initiate, those subtasks that are frequently performed for all item detail pages.

V. Predictive execution of subtasks of other subtasks

[0059] In the examples above, the prefetch service 50 was queried in response to a URL request. In some cases, it may additionally or alternatively be desirable to use the prefetch service 50 to identify subtasks to be performed preemptively as part of a particular page generation subtask. In such scenarios, the prefetch service 50 may be queried multiple times as part of a given page generation task, with different queries potentially occurring at different stages of this task.

[0060] To illustrate, assume hypothetically that the web site system 30 generates a “random detail page” that consists generally of the detail page content of three randomly selected items. To generate the random detail page, a service called “Random” is initially called to obtain three randomly selected item IDs. For each randomly selected item ID, the Catalog service is then called to obtain the associated catalog data, and the Similar Items service is called to obtain a listing of similar items. Finally, the catalog service is again called to obtain short descriptions of the similar items identified by the Similar Items service.

[0061] In this example, there would be little or no benefit to querying the prefetch service 50 when the random detail page is initially requested. This is because a very weak correlation exists between the task “random detail page” and the subtasks that will be performed to obtain the content for the randomly selected items. However, once the three randomly selected item IDs are known, the details of the remaining subtasks may be predictable. Thus, upon receiving the three item IDs from the Random service, the prefetch service 50 can be queried three times—once for each of the randomly-selected item IDs—to identify additional subtasks to be performed preemptively. The task ID passed to the prefetch service for each such query may be in the form “detail page content for <item ID>.” Special tags or instructions may be included in the template of the random detail page specifying that the prefetch service is to be called for each item ID returned by the Random service.

[0062] As illustrated by this example, the tasks identified in the task ID portion 52A of the mapping table 52 need not come from a requested URL, or even be dependent upon a URL.

VI. Non-catalog based applications

[0063] As will be apparent from the foregoing, the predictive prefetching methods described above are not limited to the generation of catalog pages, or to systems that provide access to online catalogs. Rather, the methods are generally applicable to any document generation task in which subtasks that are dependent upon the results other subtasks can be predicted based on prior document generation events.

[0064] One application for the predictive prefetch methods involves the dynamic generation of Web search result pages. For purposes of illustration, assume that the Web site system 30 implements a Web search engine that allows users to conduct general (non-site-specific) searches for web pages. Assume also that the Web search engine generates personalized Web search results pages that identify to users those search result URLs they have previously accessed. To generate such a personalized search results page, the dynamic page generator 36 initially sends the user's search query to a Web search service 44. The Web Search service 44 responds by returning a listing the URLs of web pages that are responsive to the search query, as reflected within an index generated by a web crawler program.

[0065] For each URL returned by the Web Search service 44, the dynamic page generator 36 then queries a Clickstream History service 44 to find out whether the particular user has previously accessed that URL. In this example, the task-to-subtask mapping table 52 could include a separate table entry for each of the most commonly submitted search queries, or those that satisfy some other search query bounding criteria. For example, the table entry for the task ID "results page for query = storage area networks" would specify the URLs returned the most frequently for this query, and would specify that each such URL is to be passed to the Clickstream History service together with the user's ID. This would allow some or all of the requests to the Clickstream History service to be made before the Web Search service 44 returns its results.

[0066] In this example, the service requests to the Clickstream History service may be specified in the mapping table 52 in a tokenized form, such that separate table entries are not created for specific users. Specifically, although each request to the Clickstream History service will specify a particular user ID, the mapping table 52 may specify the service

request in the form of “Clickstream History (URL, USERID),” where “URL” is an actual URL and “USERID” is a token that replaces actual user IDs. If a user having an ID of 12345 submits a search query for which such a table entry exists, the prefetch client would retrieve the subtask description, and replace “USERID” with “12345” for purposes of calling the Clickstream History service. If feedback is thereafter returned for this particular page request, the feedback message will specify the subtask in its tokenized form. Specifically, rather than indicating that the service request “Clickstream History (URL, 12345)” was made, the prefetch client 56 will indicate to the prefetch service 50 that the service request “Clickstream History (URL, USERID)” was made.

VII. Conclusion

[0067] The various inventive features described above may also be used to dynamically generate displays and display elements other than documents. For example, the invention may be used to generate a screen display, or a portion of a screen display, that contains data retrieved from multiple sources. Further, although the invention has been described primarily in the context of data retrieval subtasks, the invention may also be used, in some embodiments, to preemptively initiate subtasks that do not involve the retrieval or return of any data.

[0068] Although this invention has been described in terms of certain preferred embodiments and applications, other embodiments and applications that are apparent to those of ordinary skill in the art, including embodiments which do not provide all of the features and advantages set forth herein, are also within the scope of this invention. Accordingly, the scope of the present invention is defined only by the appended claims.